



# *Relational Database Management Systems*



## Learning Goals

*After this lecture you will be able to:*



- Understand and describe the functionality of a DBMS (Database management System)
- Describe threats to information stored in a DMBS as well as solutions to threats
- Be able to describe the functionality of a relational query language: SQL (structured query language) as well as read and write small part of SQL-queries

## DBMS and RDBMS



- A DataBase Management System (DBMS) is a set of programs that help users manage large sets of data in an easy and consistent way. By manage is meant that the users shall be able to DEFINE, INSERT, UPDATE, DELETE AND RETRIEVE data in a consistent and secure manner.
- A Relational DataBase Management System (RDBMS) is a DataBase Management System where the data is structured according to the relational model.

## *Architecture of a Database Management System*

*An DBMS is one type of Information Management system, often referred to as a Transaction Management System. It interacts with its users, applications outside of the DBMS, and the database itself to capture and analyze data.*

**Presentation**

(Graphical) user interfaces

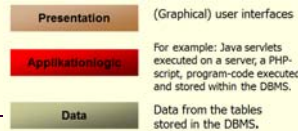
**Applikationlogic**

For example: Java servlets executed on a server, a PHP-script, or even program-code executed and stored within the DBMS.

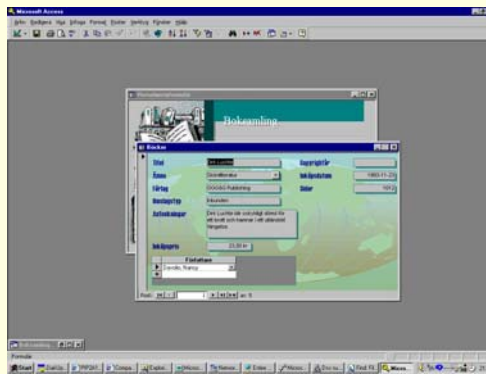
**Data**

Data from the tables stored in the DBMS.

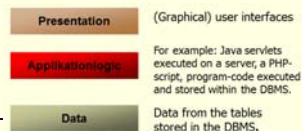
# Presentation



How does (end-) users retrieve information from the DBMS? Graphical user interfaces to various applications provide information about data stored in the database management system. User interfaces may be implemented within the DBMS or be a program outside of the DBMS that connects to the DBMS via some protocol-standard.



# Query-languages



- How does application programs retrieve data from the database management system?
- Through a query language.

```
SELECT ArtName  
FROM Artefact  
WHERE Artist = "Picasso"
```

## Query-languages: *SQL- Structured Query Language*

---

- SQL has several parts, **DDL, DML, DCL and TCL**
- Through **DDL**, Data **D**efinition Language, we **define** tables, rules etc. (*see the former lecture*)
- Using **DML**, Data **M**anipulation Language, we can then write queries against the tables we created
- **DCL**, Data **C**ontrol Language, deals with the rights, permissions and other controls of the database system (*more on this on transactions in DBMS:s*)
- **TCL**, Transaction **C**ontrol Language, TCL commands deals with the transaction concept in the DBMS (*more on this on transactions in DBMS:s*)

7

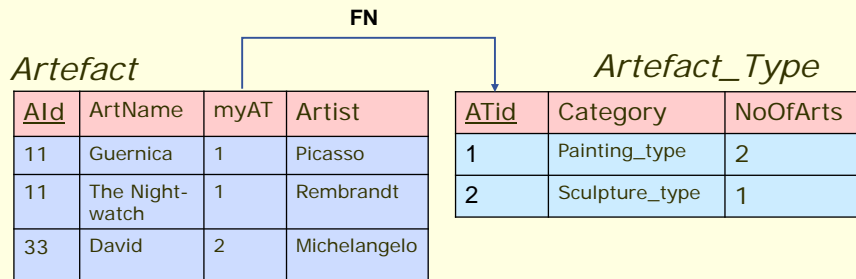
## Query-languages: *SQL- DML*

---

- Through **DML**, Data **M**anipulation Language, we can write **queries** against the tables in the DMBS
- **SELECT** – is used to retrieve data from the database.
- **INSERT** – is used to insert data into a table.
- **UPDATE** – is used to update existing data within a table.
- **DELETE** – is used to delete rows from a table.

8

## DML: example tables



9

## Query-languages: SQL DML: An example DML Query:



SELECT ArtName  
FROM Artefact  
WHERE Artist = "Picasso"

In the SELECT-clause **columns** are chosen to be displayed

The FROM-clause tells what **tables** are going to be searched

The WHERE-clause contains **conditions** that shall hold for the rows to be displayed

<u>ArtName</u>
Guernica

The **result!**

10

## Query-languages: DML: Join-query

Artefact				Artefact_Type		
Aid	ArtName	myAT	Artist	ATid	Category	NoOfArts
11	Guernica	1	Picasso	1	Painting_type	2
11	The Night-watch	1	Rembrandt	2	Sculpture_type	1
33	David	2	Michelangelo			

Give me the names and artists of all paintings!

```
SELECT ArtName, Artist
FROM Artefact, Artefact_Type
WHERE myAT = ATid
AND Category = "Painting_type"
```

In the SELECT-clause **columns** are chosen to be displayed

The FROM-clause tells what **tables** are going to be searched

The WHERE-clause contains **conditions** that shall hold for the rows to be displayed. Here the WHERE-clause contains **TWO** conditions!

ArtName
Guernica
The Night-watch

The **result!**

11

## Query-languages: DML INSERT

Artefact

Aid	ArtName	myAT	Artist
11	Guernica	1	Picasso
11	The Night-watch	1	Rembrandt
33	David	2	Michelangelo

```
INSERT INTO Artefact(44, 'Mona Lisa', 1, 'Da Vinci')
```

Artefact

Aid	ArtName	myAT	Artist
11	Guernica	1	Picasso
11	The Night-watch	1	Rembrandt
33	David	2	Michelangelo
44	Mona Lisa	1	Da Vinci

12

# Transaction management

## - Transactions and ACID



- What is a transaction?
- An action, or set of actions!, that reads and/or writes the content of the database. It is performed by a user or an application program. It is defined as ONE unit that either happens as a whole, or not at all!
- A transaction is performed with respect to a set of very important rules/properties referred to as ACID (to be introduced in the next slide).

## Transaction principles - ACID

- **Atomicity** – "All or nothing", i.e. either the entire action, or actions, happens (a COMMIT is signaled if all is well) or nothing happens (ROLLBACK if something went wrong during the execution of the transaction)
- **Consistency** – Database integrity shall hold before and after (not necessarily during) the transaction, for instance entity- and referential integrity
- **Isolation** – all transaction must be independent of each other (for instance, two transaction must not "over-write" the same data)
- **Durability** – updates shall be permanent after a COMMIT, e.g. even if there is a database-crash

# Transactions and concurrency

**PROBLEM:** • Transactions often demand access to the SAME resource (table/row/cell)

- Several transactions shall (ideally) be able to be run in parallel ("concurrent/at the same time")
  - At least this is how the users want to perceive it

**versus**

- The transactions shall be independent of each other, reads they shall not "disturb" or "interfere" with each other.

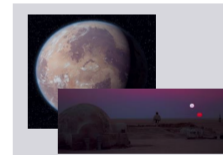
# Concurrency and Isolation Example

## Transaction Management



- The phantom problem

Transaktion A	Transaktion B
Hämta alla livsformer från Tatooine	
	Ta bort Luke (som flyttat från Tatooine)
...men Luke är fortfarande med här	
Transaktion C	Transaktion D
Hämta alla livsformer från Tatooine	
	Lägg till Luke (som flyttat tillbaka till Tatooine)
...men här kommer Luke inte med	



Bilderna från Wikipedia



## Transactions and concurrency

### **Solutions:**

- *Run everything serially only (non parallel): not realistic, long wait-times*
- *Conservative parallel techniques, also know as "pessimistic techniques" : point of departure is that conflicts will arise and these are solved via **LOCKS, TIMESTAMPS.***

## Transactions and concurrency

### **LOCKS come in two main variants:**

- **Read lock (shared lock) on a resource (-s)**
  - » *can be given to several transactions wrt the same resource*
- **Write lock (exclusive lock) on a resource (-s)**
  - » *the resource may not even be read by other transactions*
- **Locks are given/released wrt need and type of lock-schema**
  - » *Granularity of locks: usually database, table, row, cell. Important not to lock more resources than necessary.*

# Security, Threats and Solutions

## Security and **THREAT**

- *Deliberate*
- *Non deliberate*
- *May effect the DBMS*
- *Or the whole organization*

# Security, **Threats** and Solutions

Threats may be directed against:

### Hardware

- Servers
- Networks

### Software

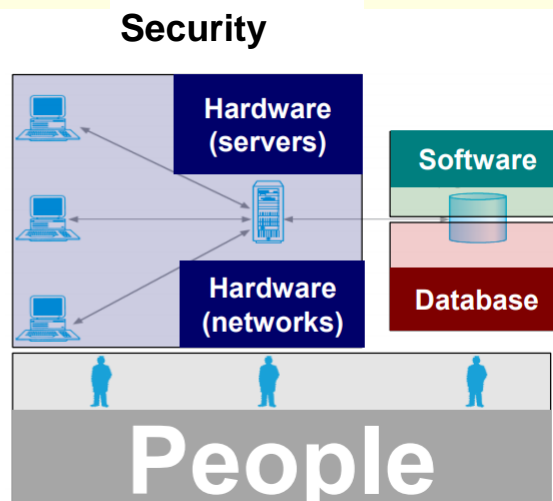
- DBMS
- Application software

### Database

- Data

### People

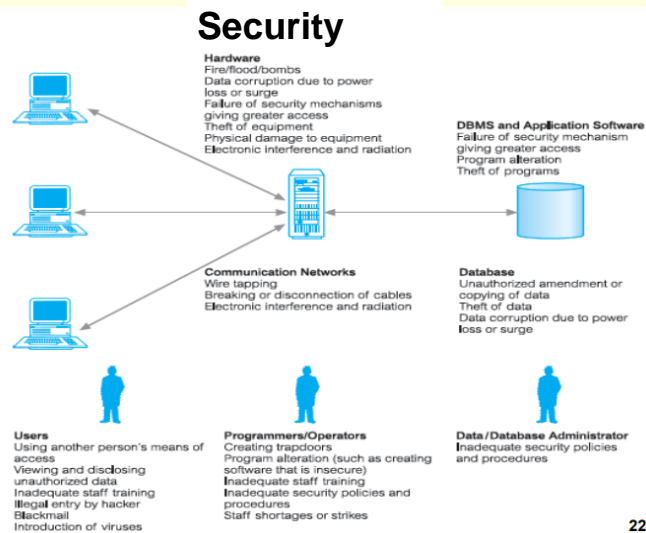
- DA/DBA
- Programmers
- Operators
- End-users



# Security, Threats and Solutions

## Consequences of threats :

- Hardware faults
- Software bugs
- Theft
- Fraud
- Diminished:
  - accessibility
  - personal integrity
  - data integrity



## Solutions

- *Authorization* (grant user access only to certain resources)
- *Authentication* (demand identification via userIDs, pin-codes, retina control, finger-prints etc.)
- *Backup / recovery* (redundance)
- *Integrity rules*
- *Views*
- *Triggers*

## Solution examples: SQL DCL

**Authorization** (grant user access only to certain resources) via

**DCL(Data Control Language)** : DCL includes commands such as GRANT and REVOKE which mainly deals with the rights, permissions and other controls of the database system.

**Examples of DCL commands:**

**GRANT**-gives user's access privileges to database.

**REVOKE**-withdraw user's access privileges given by using the GRANT command.

## Solution examples: SQL DCL cont.

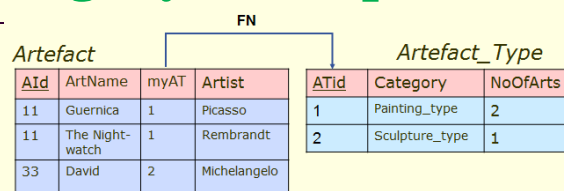
```
GRANT privilegier  
ON objekt  
TO subjekt  
[WITH GRANT OPTION]
```

- *Privileges – SELECT, INSERT, UPDATE, DELETE, REFERENCES, ALL PRIVILEGES*
- *Object : Table , view, domain, etc*
- *Subject : User, group-of-users, role (DBA etc.), PUBLIC*
- *WITH GRANT OPTION – Allows the subject to, in turn, grant the privilege to another subject*

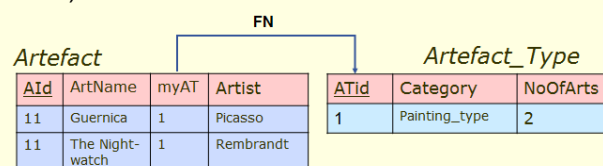
## Solution examples: DBMS integrity rules (part of DDL)

- Entity integrity – primary keys may not be NULL or contain duplicates
- Referential integrity – foreign keys must have a value corresponding to the referenced primary key, OR be NULL
- Foreign key rules – what happens to the FK-values when the PK changes (i.e. ON UPDATE or ON DELETE)? » CASCADE » SET NULL / NULLIFY » RESTRICT / NO ACTION

## Solution examples: DBMS integrity rules (part of DDL)



CREATE TABLE Artefact(Aid String NOT NULL, ArtName String NOT NULL, MyAT Integer NOT NULL, Artist String NOT NULL, PRIMARY KEY (Aid), FOREIGN KEY (MyAT) REFERENCES Artefact\_Type(ATName) ON DELETE CASCADE ON UPDATE CASCADE)



## Solution examples: Views

The database may be accessed through VIEWS – i.e. a set of retrieved tables/rows/part\_of rows that can be tailored towards different users with *different information needs*. A view is defined on top of one or several tables via a CREATE VIEW command that contains a SELECT-query.

Course	Student	Result	Group
IS:4	Kalle	VG	A
IS:4	Pelle	G	A
IS:4	Olle	G	B
IS:4	Lisa	U	B
2i1033	Kalle	5	1
2i1033	Lisa	4	1
2i1033	Oskar	3	2
2i1033	Eskil	U	2

...  
The Scheduler does only want to read the columns Course, Student and Group.

The students are only allowed to read the columns Student and Course.

Teachers and head-master need to be able to read and write the entire table.

27

## Solution examples: Views cont.

Views are defined on one or several base-tables (or even other views):

```
CREATE VIEW Student_view AS
(SELECT Course, Student
 FROM COURSE_OCCASION)
```

```
CREATE VIEW Teacher_view AS
(SELECT * FROM COURSE_OCCASION)
```

```
CREATE VIEW Scheduler_view AS
(SELECT Course, Student, Group
 FROM COURSE_OCCASION)
```

The Scheduler does only want to read the columns Course, Student and Group.

The students are only allowed to read the columns Student and Course.

Teachers and head-master need to be able to read and write the entire table.

28

## Solution examples

- **VIEW WITH CHECK OPTION**

- Data can be INSERT:ed through the view The INSERT:s must follow the rules of the view

```
CREATE VIEW Fanskap AS
SELECT regNo, namn, fanskapsstatus
FROM Person
WHERE fanskapsstatus IN ('Boden Hockey', 'AIK', 'Man United')
WITH CHECK OPTION
```

```
INSERT INTO Fanskap
VALUES ('b2345', 'Thelemyr', 'Boden Hockey')
```

OK

```
INSERT INTO Fanskap
VALUES ('g4567', 'Wenger', 'Arsenal')
```

EJ OK



## Solution exempel: Trigger

- An **event** that triggers (INSERT, UPDATE, DELETE, ...)
- An **affected object**, usually a table or a column
- A **condition** that must be true for the trigger body to be executed
- A **trigger body** executed when the condition is true

*By using triggers it is possible to implement business rules that cannot be implemented otherwise (for instance using classes, domain rules etc.): - credit limit must not be exceeded, - reported mileage must be higher than earlier reported mileage, - updating calculated attributes: no of employees in a company must correspond with the number of tuples having that company as the value for the foreign key.*



## Solution example: Trigger cont.


*Artefact*

<u>AId</u>	ArtName	myAT	Artist
11	Guernica	1	Picasso
11	The Night-watch	1	Rembrandt
33	David	2	Michelangelo

INSERT INTO Artefact(44, 'Mona Lisa', 1, 'Da Vinci')

*Artefact*

<u>AId</u>	ArtName	myAT	Artist
11	Guernica	1	Picasso
11	The Night-watch	1	Rembrandt
33	David	2	Michelangelo
44	Mona Lisa	1	Da Vinci



## Solution: Triggers Example


FN

*Artefact*

<u>AId</u>	ArtName	myAT	Artist
11	Guernica	1	Picasso
11	The Night-watch	1	Rembrandt
33	David	2	Michelangelo
44	Mona Lisa	1	Da Vinci

*Artefact\_Type*

<u>ATid</u>	Category	NoOfArts
1	Painting_type	2
2	Sculpture_type	1



Not correct!





## Solution: Triggers Example (DB2)

```
CREATE TRIGGER increment_noOfArts
AFTER INSERT ON Artefact ← event!
FOR EACH ROW
REFERENCING NEW as n
BEGIN
UPDATE Artefact_Type ← affected object!
SET NoOfArts = NoOfArts + 1 ← trigger body!
WHERE n.myAT = ATId; ← condition!
END@
```

FN

Artefact				Artefact_Type		
Aid	ArtName	myAT	Artist	ATId	Category	NoOfArts
11	Guernica	1	Picasso	1	Painting_type	3
11	The Night-watch	1	Rembrandt	2	Sculpture_type	1
33	David	2	Michelangelo			
44	Mona Lisa	1	Da Vinci			